MVP ve Mediator ile Loose Coupled, Modüler UI Geliştirme

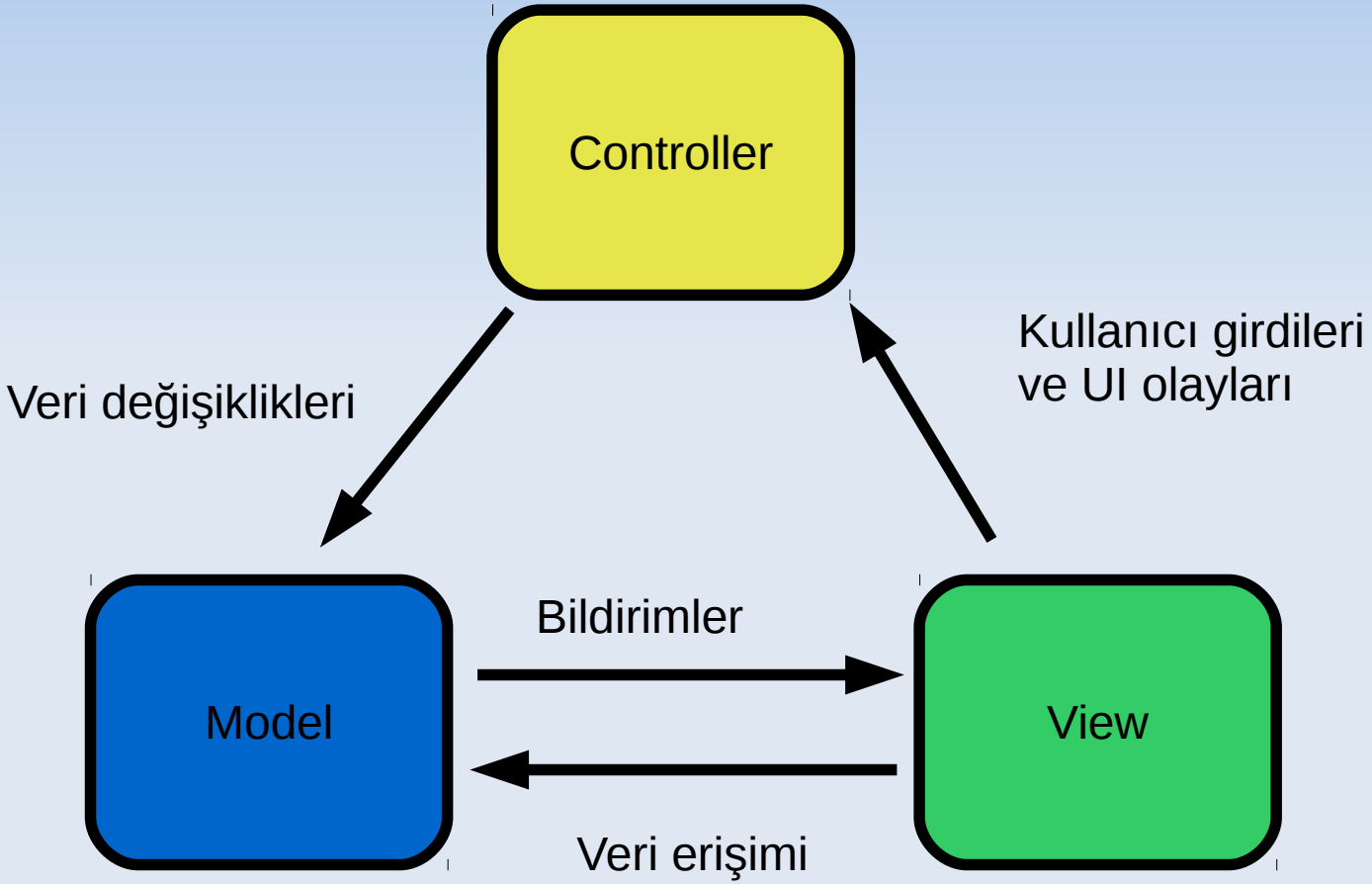
Hakkımızda

- Kenan Sevindik, 1999 ODTÜ mezunu
- 15 yıllık kurumsal uygulama geliştirme tecrübesine sahip
- Uzmanlık ve ilgi alanları
  - Kurumsal uygulama mimarileri, middleware servisler
  - OOP ve AOP, Design Patterns
  - Enterprise Java, Vaadin, Spring, Spring Security, Hibernate
- 2011 yılında Harezmi Bilişim Çözümleri'ni kurdu

Hizmetlerimiz

- Kurumsal uygulama geliştirme faaliyetleri yürütüyoruz
- Danışmanlık ve koçluk hizmetleri veriyoruz
- Kurumsal Java eğitimleri düzenliyoruz
  - Java Programlama Dili Eğitimi
  - Spring Application Framework Eğitimi
  - Spring Security ile Web Uygulama Güvenliği Eğitimi
  - AspectJ ve Spring AOP ile AOP Eğitimi
  - Hibernate Persistence Framework Eğitimi
  - Vaadin ile RIA Eğitimi
  - Design Patterns Eğitimi

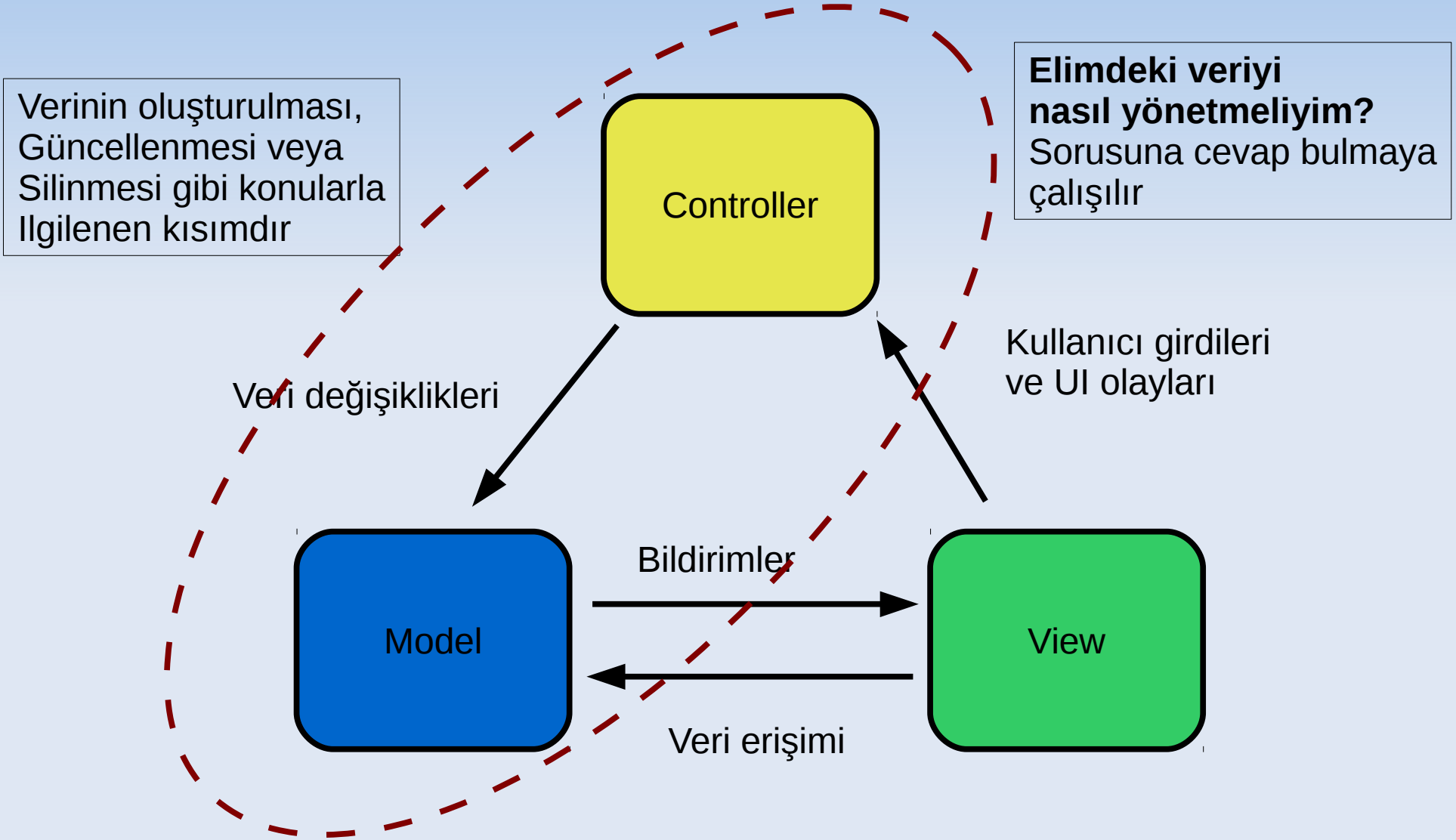
Mimarisel Bir Örüntü: MVC



MVC'nin Temel İşlevi

“**Seperation of Concern**”

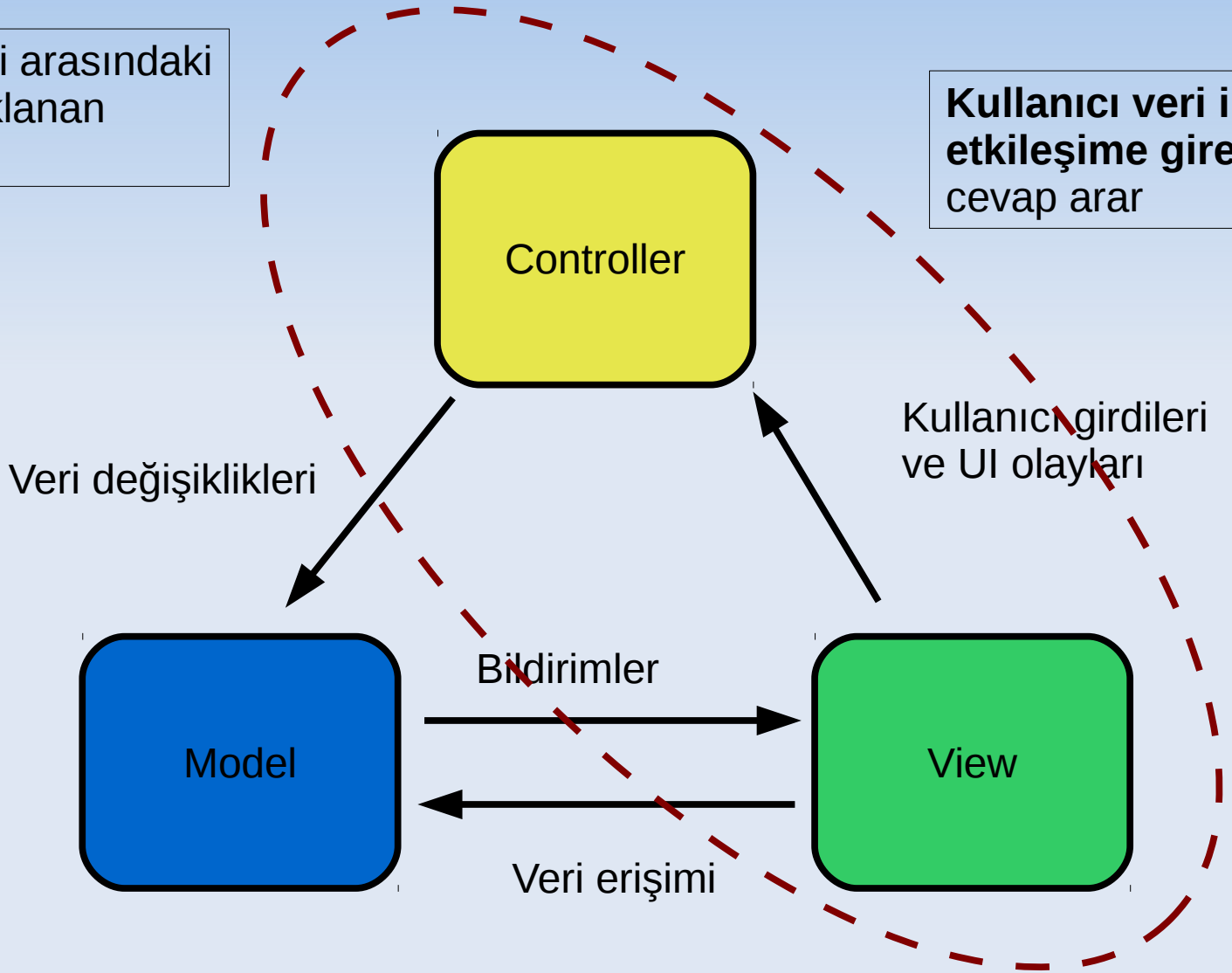
MVC ve Veri Yönetimi



MVC ve Kullanıcı Arayüzü Etkileşimi

Kullanıcı ile veri arasındaki etkileşime odaklanan kısımdır

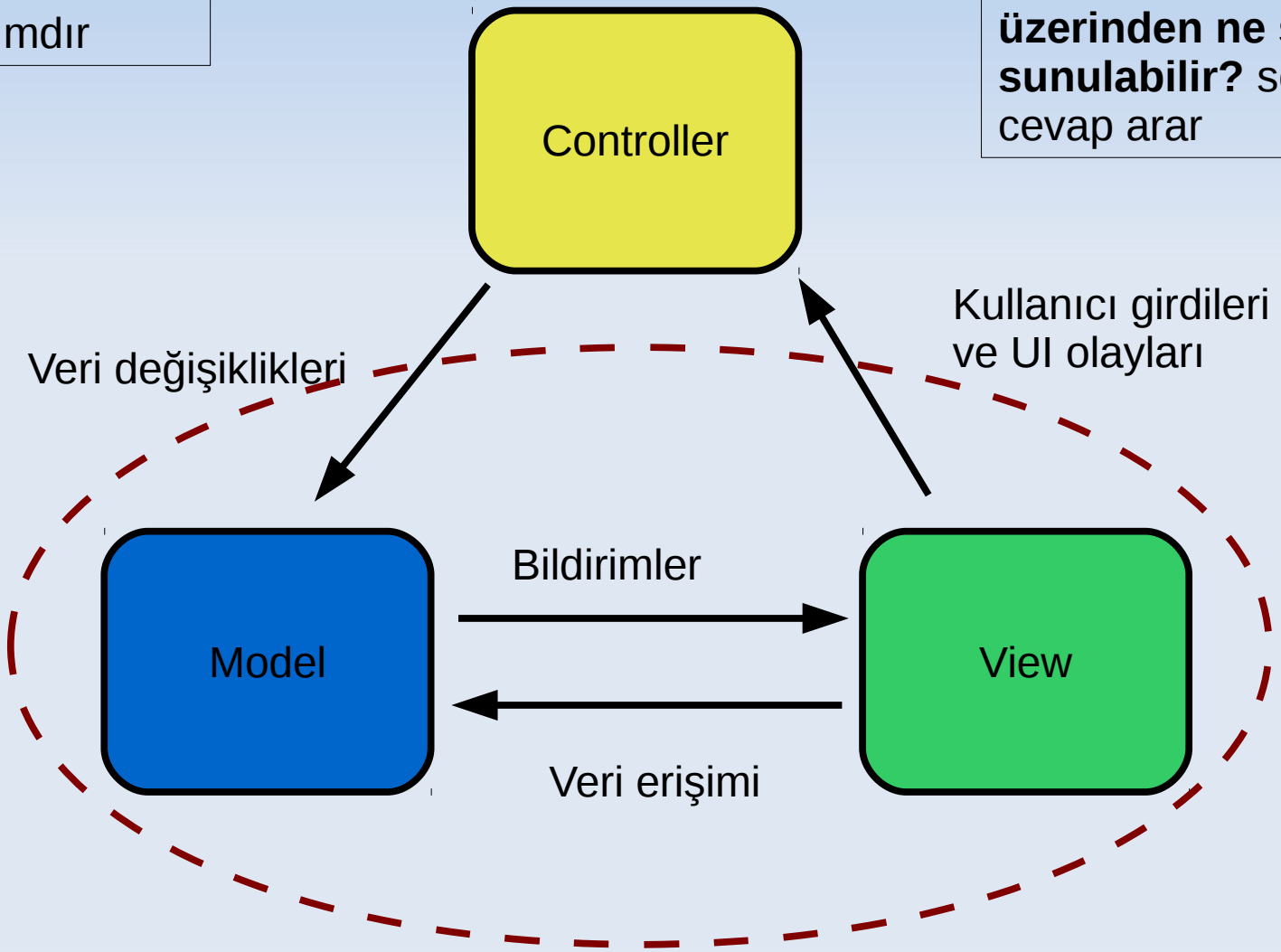
Kullanıcı veri ile nasıl etkileşime girer? sorusuna cevap arar



MVC ve Verinin Gösterimi

Verinin kullanıcı arayüzü üzerinden gösterimine odaklanan kısımdır

Veri kullanıcı arayüzü üzerinden ne şekilde sunulabilir? sorusuna cevap arar



Model View Presenter

View

UI event'leri uygulamaya özel event'lere dönüştürülür

Presenter

UI üzerindeki değişiklikler Presenter tarafından yansıtılır

Presenter Model üzerinde Değişiklik Yapabilir Model verisine erişebilir

Model

Model üzerindeki Değişiklikler Event'ler Ile Presenter'a iletilir

Nereden Başlamalı?

Nasıl Kodlamalı?

Presenter'dan

Çünkü...
kullanıcı senaryoları ve fonksiyonel gereksinimler bire bir Presenter içindeki fonksiyonlara karşılık gelmektedir

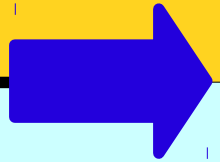
Çünkü...

Geliştiriciler, Presenter kısımlarını kodlamaya odaklanabilirler

Kendi içlerinde de fonksiyonel gereksinimlere göre gruplara ayrılarak paralel çalışabilirler

Presenter Kodları

Fonksiyonel Gereksinimler



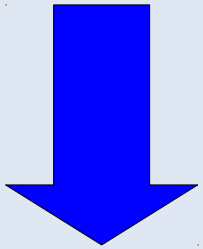
View Arayüzleri

View Implementasyonları

UI geliştiriciler ise tamamen GUI geliştirmeye odaklanabilirler. View implemantasyonu içerisinde sadece UI widget'ların oluşturulması, sayfalara yerleştirilmesi söz konusudur.

Çünkü...

Geliştirme sürecinde **TDD yaklaşımına uygun** çalışmaya olanak sağlamaktadır

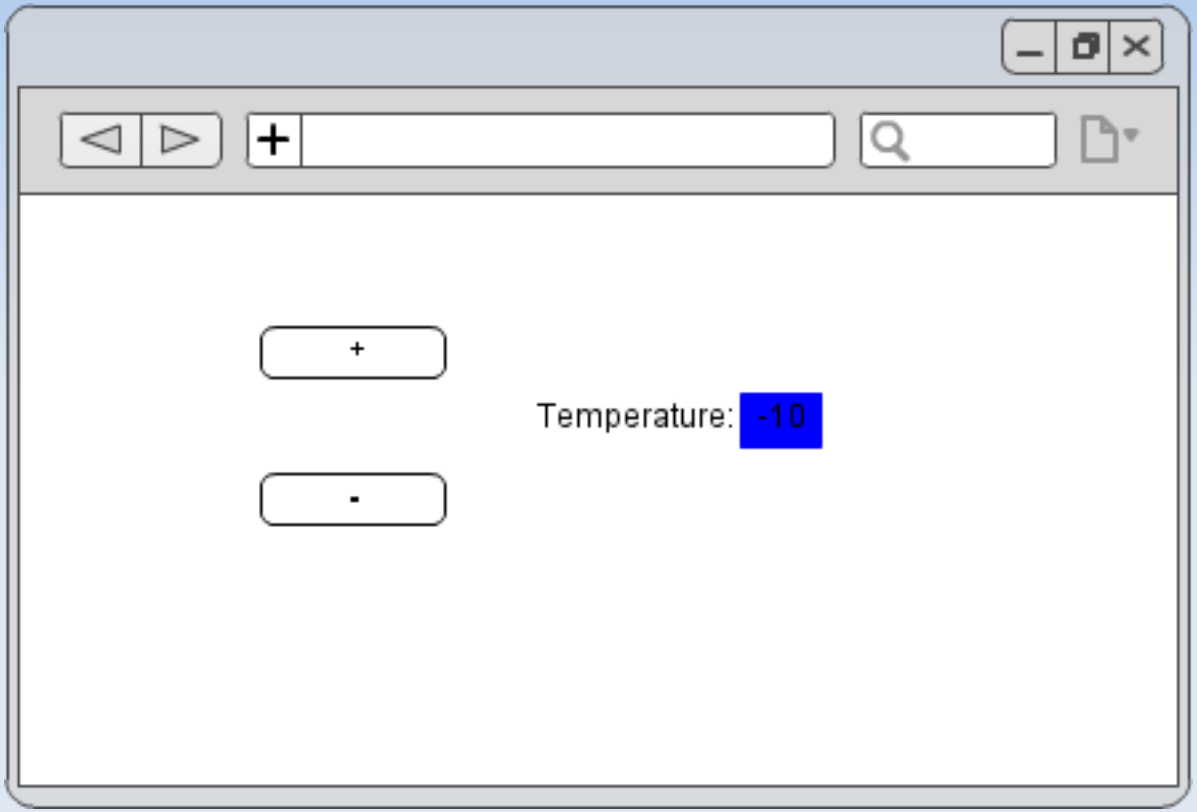


View, Model ve ihtiyaç duyulan servis bileşenleri mock'lanarak Presenter'a verilir

Örnek 1

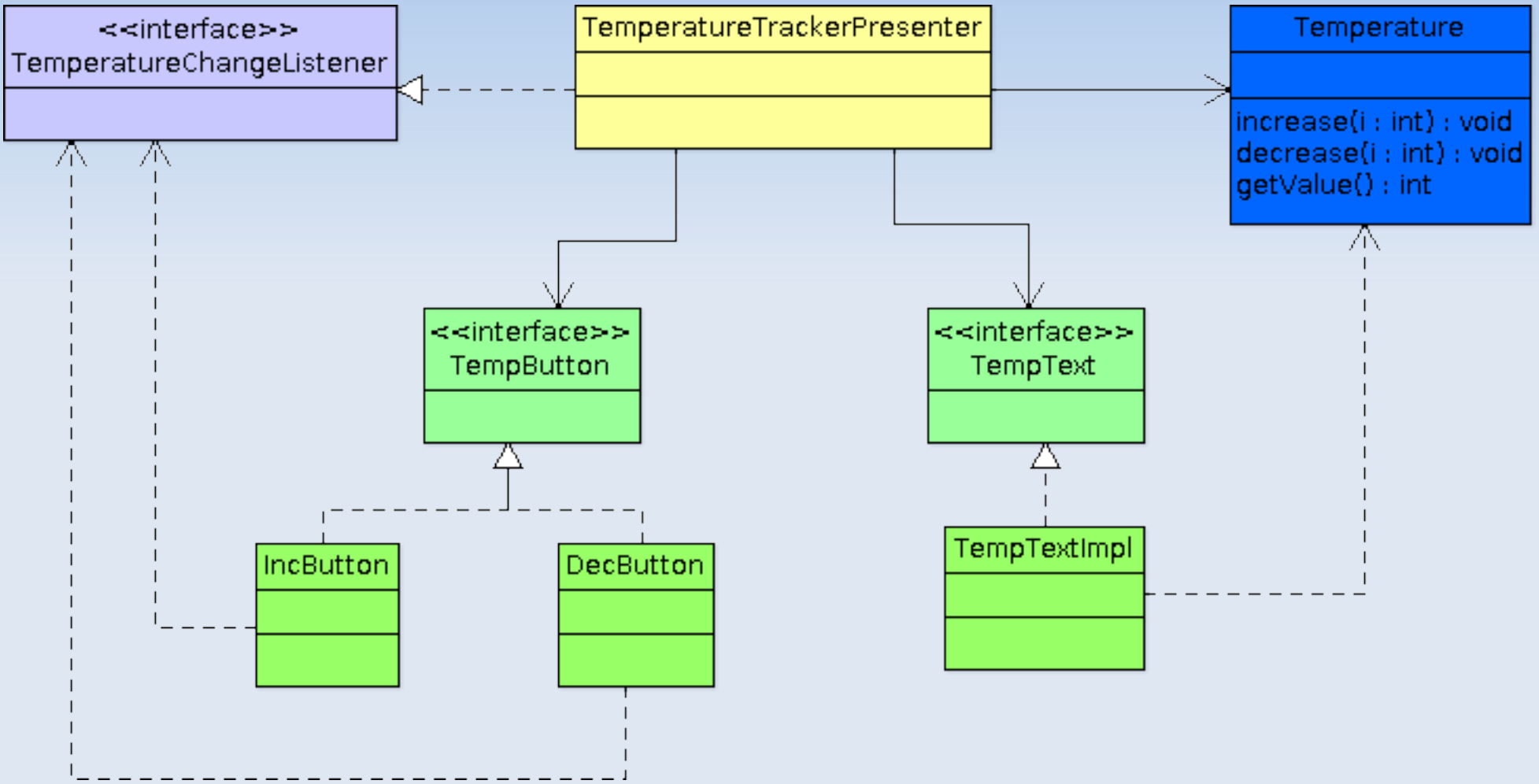
- Kullanıcımız sıcaklık değerini birer birer artırıp düşürdüğü bir gösterge istiyor.
- Sıcaklık değerini birer birer artırdığı veya azalttığı buton'lar olacak.
- Sıcaklık artışı ve azalışı anında ekrana sayısal olarak yansıtılacak.
- Sıcaklığa göre göstergedeki değerin arka planı dinamik olarak renk değişecek.

UI Mockup



Kullanıcı + ve – butonlarına tıklayarak sıcaklık değerini birer birer artırıp düşürebilecek. Sıcaklık değeri 0'ın altında iken arka plan mavi, sıfırın üstünde yeşil, 20 derece'nin üstünde sarı, 40 derecenin üstünde de kırmızı olacak.

Sınıf ve Arayüzler



Birim Testleri

```
@Mock
private TempButton incButton;

@Mock
private TempButton decButton;

@Mock
private TempText tempText;

@Mock
private TemperatureChangeEvent event;

private Temperature temperature;

private TemperatureTrackerPresenter presenter;
```

Birim Testleri

```
@Test
public void textShouldBeUpdatedWhenTemperatureChanges() {
    Mockito.when(event.getChange()).thenReturn(1);

    Mockito.when(event.getType()).thenReturn(Change.INCREASE);

    Assert.assertEquals(0, temperature.getValue());

    presenter.temperatureChanged(event);


    Assert.assertEquals(1, temperature.getValue());

    Mockito.verify(tempText).refresh();
}
```

Birim Testleri

```
@Test
public void colorShoulBeRedWhenTemperatureAboveForty() {
    Mockito.when(event.getChange()).thenReturn(41);

    Mockito.when(event.getType()).thenReturn(Change.INCREASE);

    Assert.assertEquals(0, temperature.getValue());

    presenter.temperatureChanged(event);

    Assert.assertEquals(41, temperature.getValue());

    Mockito.verify(tempText).red();

    Mockito.verify(tempText).refresh();
}
```

Presenter

```
public void temperatureChanged(TemperatureChangeEvent event) {
    int value = event.getChange();

    if(event.getType() == Change.INCREASE) {
        temperature.increase(value);
    } else {
        temperature.decrease(value);
    }

    value = temperature.getValue();

    if(value < 0) {
        tempText.blue();
    } else if (value > 0 && value < 24) {
        tempText.green();
    } else if (value > 24 && value < 40) {
        tempText.yellow();
    } else if (value > 40) {
        tempText.red();
    }

    tempText.refresh();
}
```

IncButton View

```
public void addTemperatureChangeEventListener(
    TemperatureChangeListener changeListener) {
    listeners.add(changeListener);
}

public IncButton() {
    Button btn = new Button("+");
    btn.addListener(this);
    setCompositionRoot(btn);
}

@Override
public void buttonClick(ClickEvent event) {
    TemperatureChangeEvent temperatureChangeEvent = new
                  TemperatureChangeEvent(1, Change.INCREASE);

    for(TemperatureChangeListener listener:listeners) {
        listener.temperatureChanged(temperatureChangeEvent);
    }
}
```

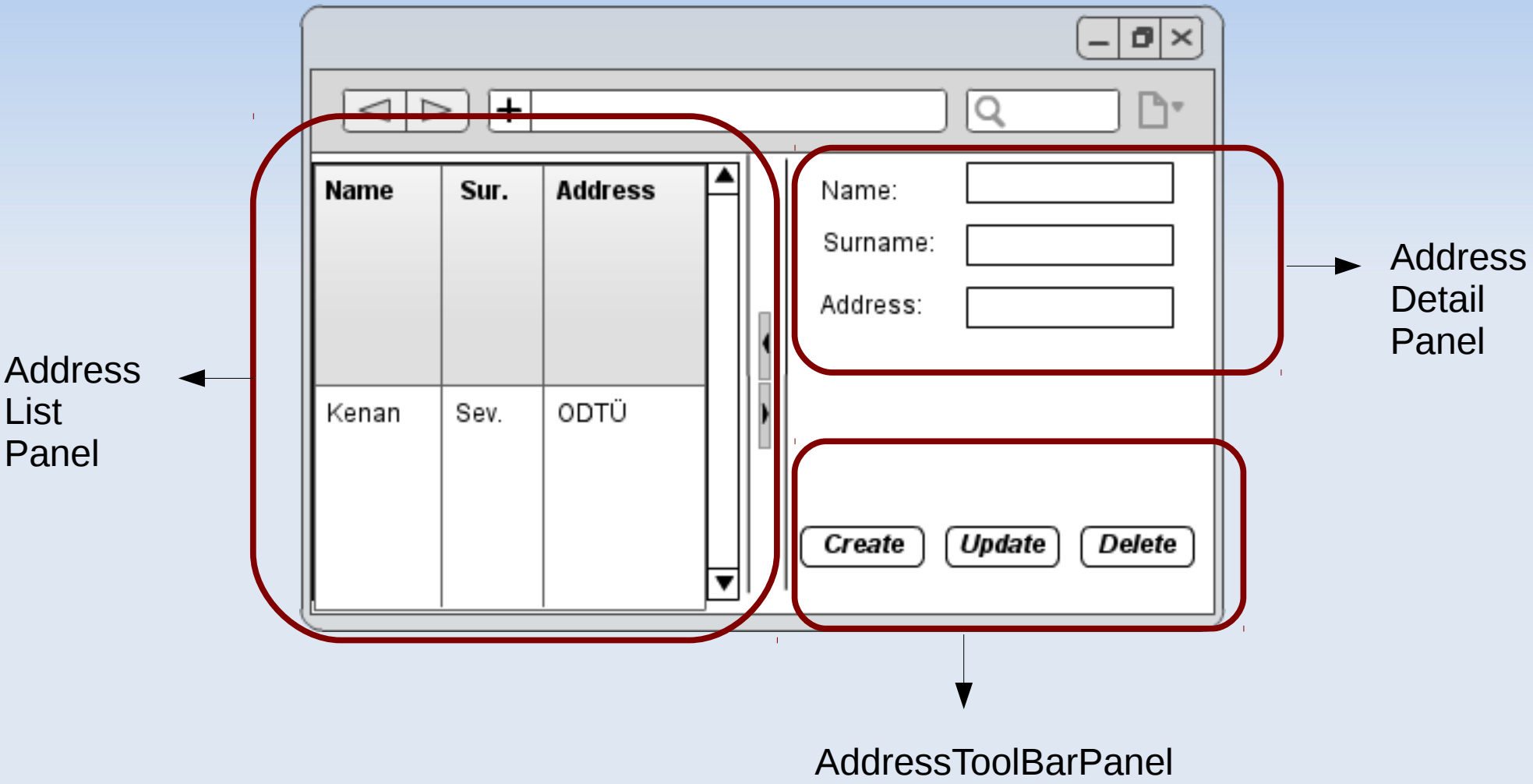
TempText View

```
private Label label;

public TempTextImpl(Temperature temperature) {
   HorizontalLayout ho = new HorizontalLayout();

   ho.setSpacing(true);
   ho.addComponentAsFirst(new Label("Temperature :"));
   label = new Label(new MethodProperty(temperature,
"value"));
   ho.addComponent(label);

   setCompositionRoot(ho);
}

@Override
public void blue() {
   label.setStyleName("bluelabel");
}
```

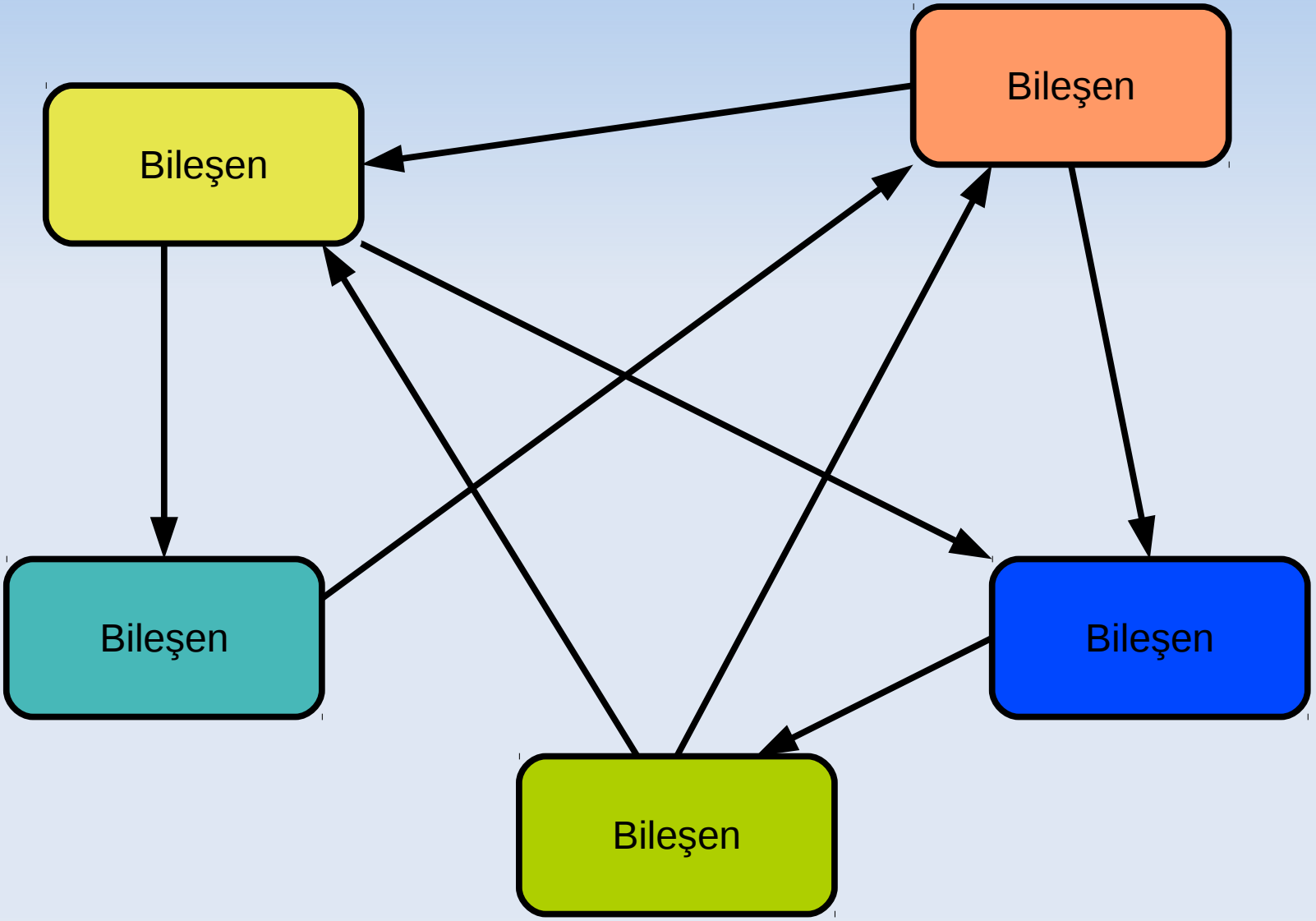
Örnek 2

- Kullanıcımız adres bilgilerini yönettiği bir uygulama istemektedir.
- Adres bilgileri sol tarafta bir liste içerisinde listelenecektir.
- Listelenen adres kayıtlarından herhangi biri seçildiği vakit sağ taraftaki detay ekranında görüntülenecektir.
- Kayıt seçili değilken toolbar sadece Create butonu aktif iken, kayıt seçildiği zaman Create butonu pasif, Update ve Delete butonları ise aktif olacaktır.
- Bu kayıt üzerinde herhangi bir değişiklik kaydedildiği vakit değişiklikler hemen sol taraftaki listeye yansıtılacaktır.

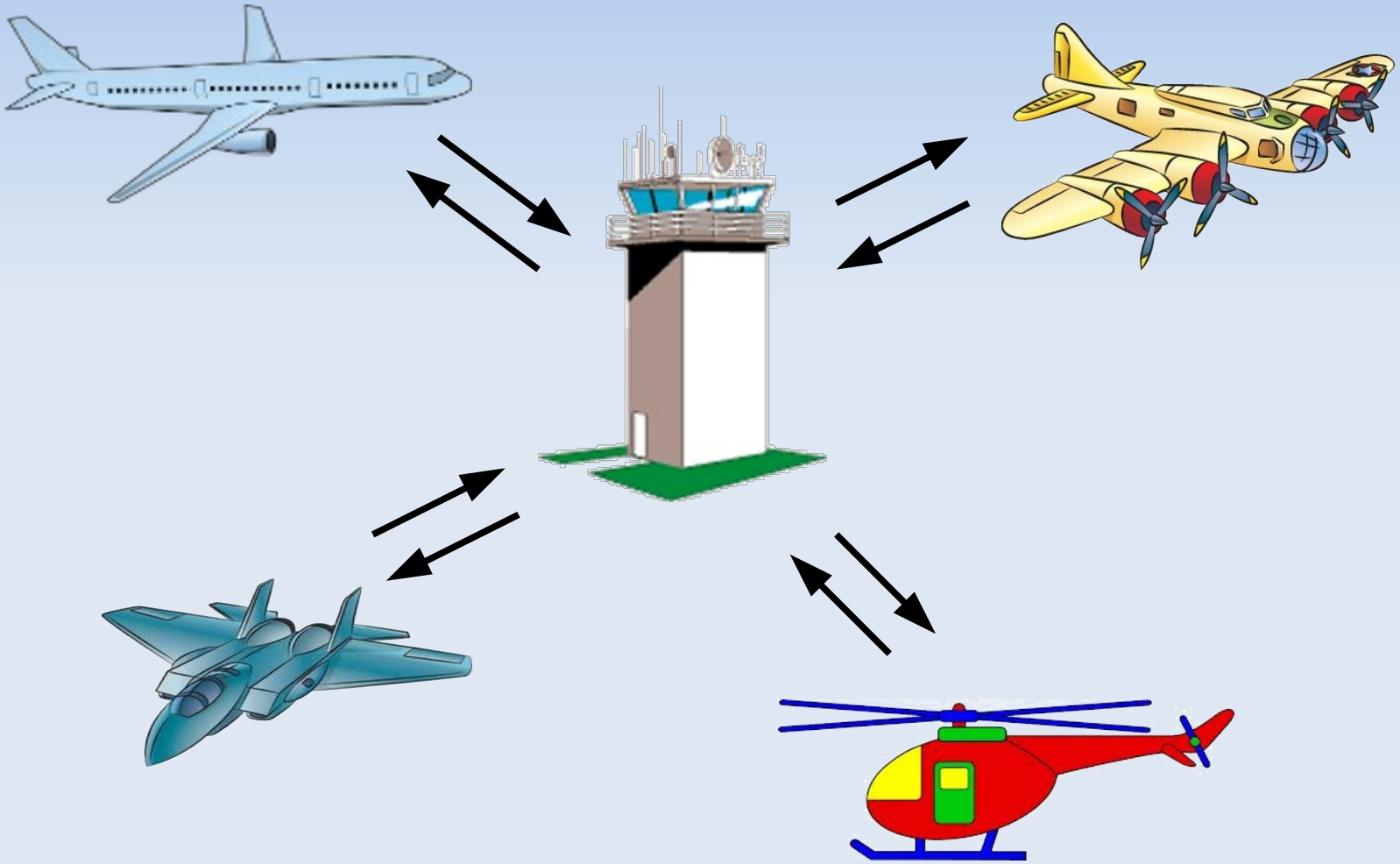
UI Mockup


Name
Sur.
Address
Kenan
Sev.
ODTÜ
Name:
Surname:
Address:
Create
Update
Delete
Address List Panel
Address Detail Panel
AddressToolBarPanel

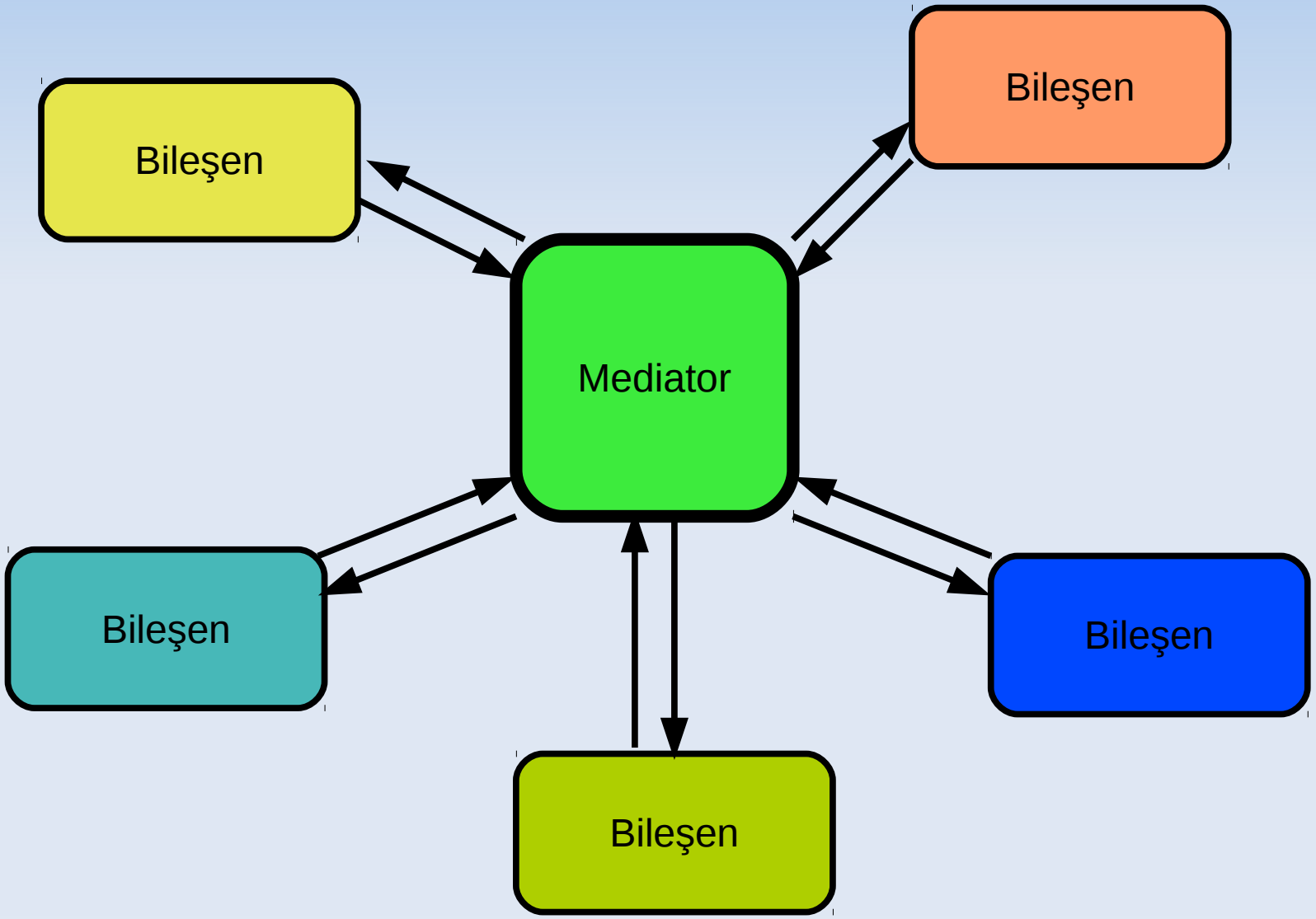
Bileşenler Arasındaki Etkileşim

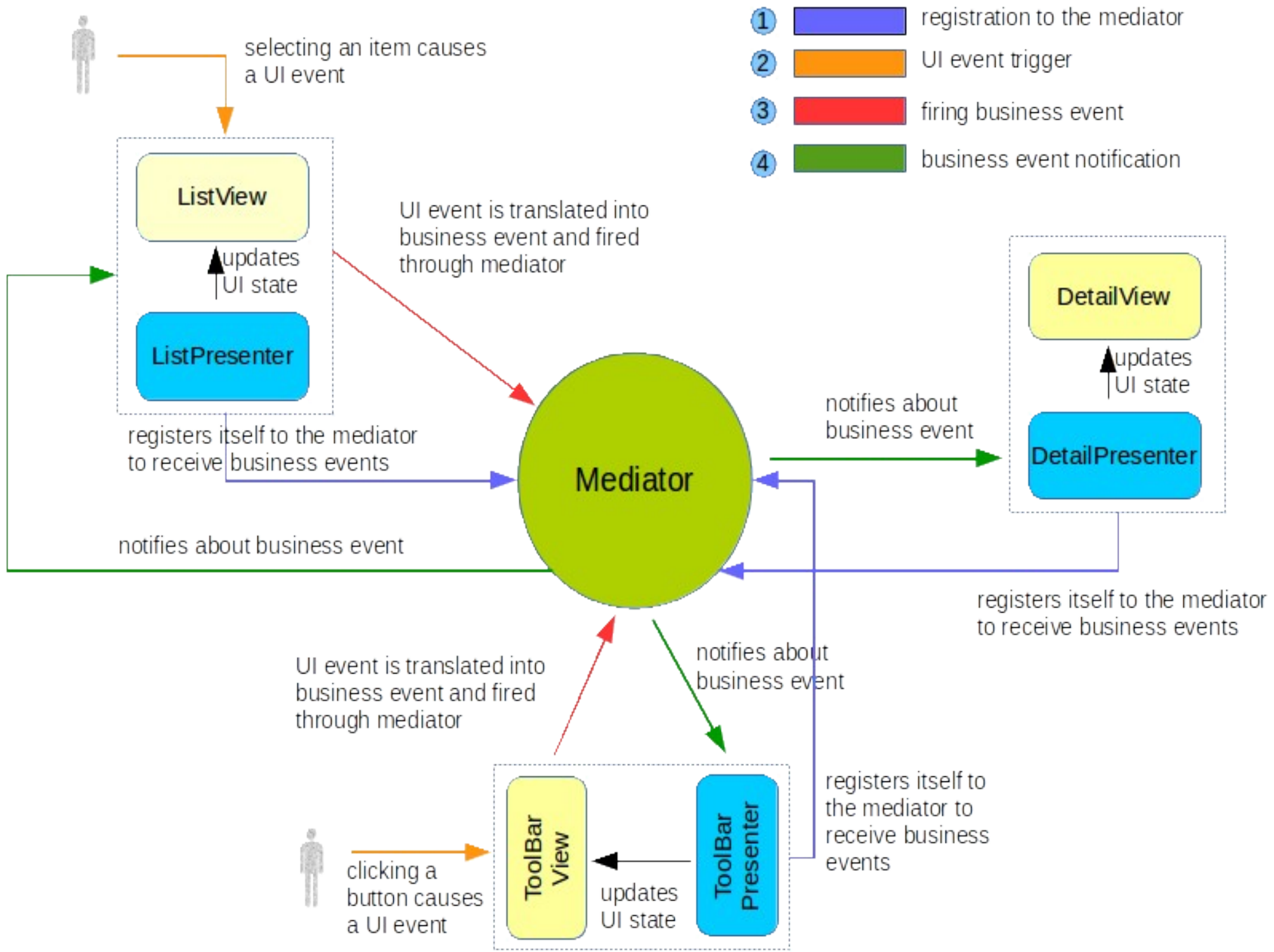


Mediator Pattern

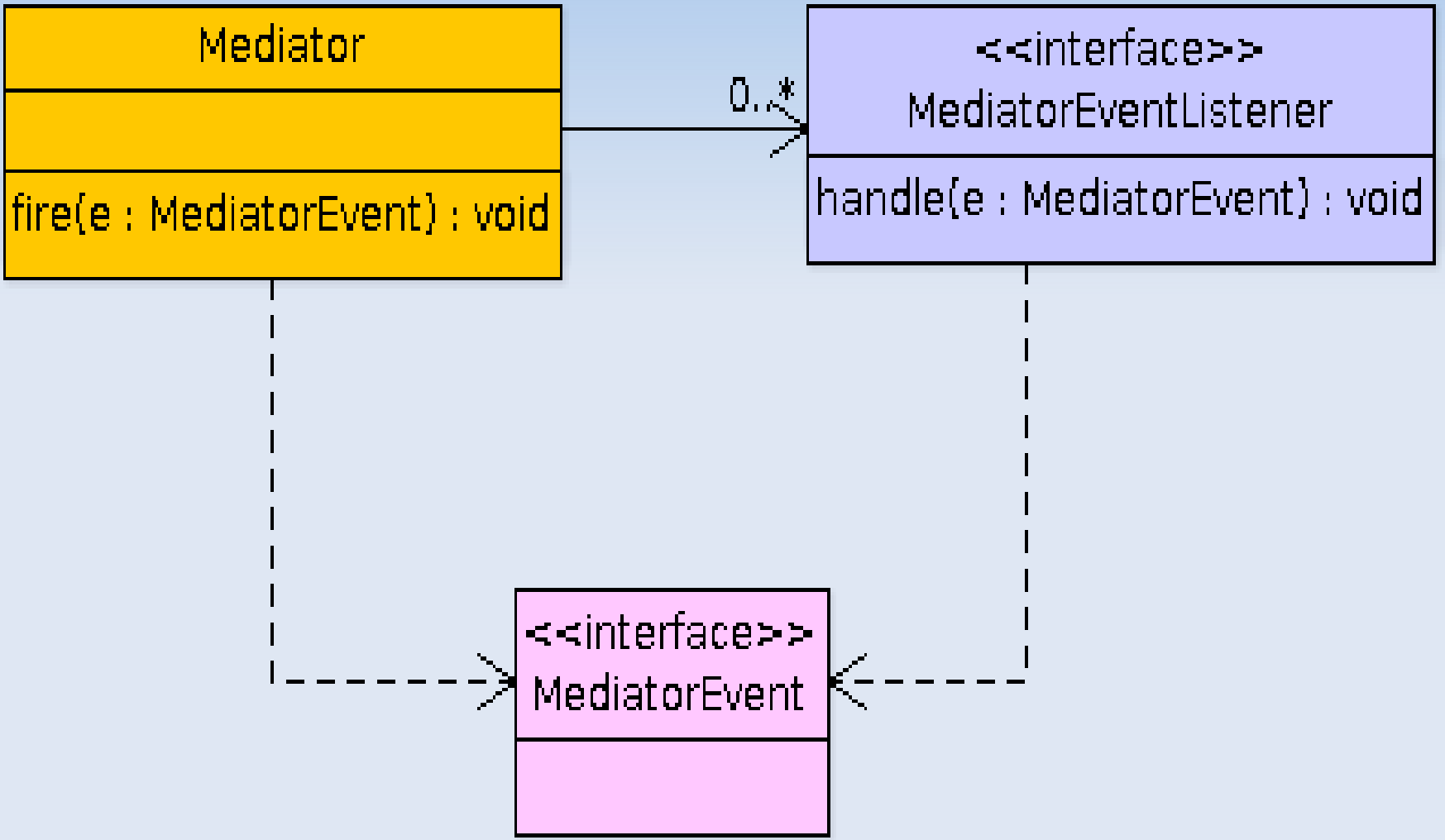


Mediator Sonrası Bileşenler Arasındaki Etkileşim

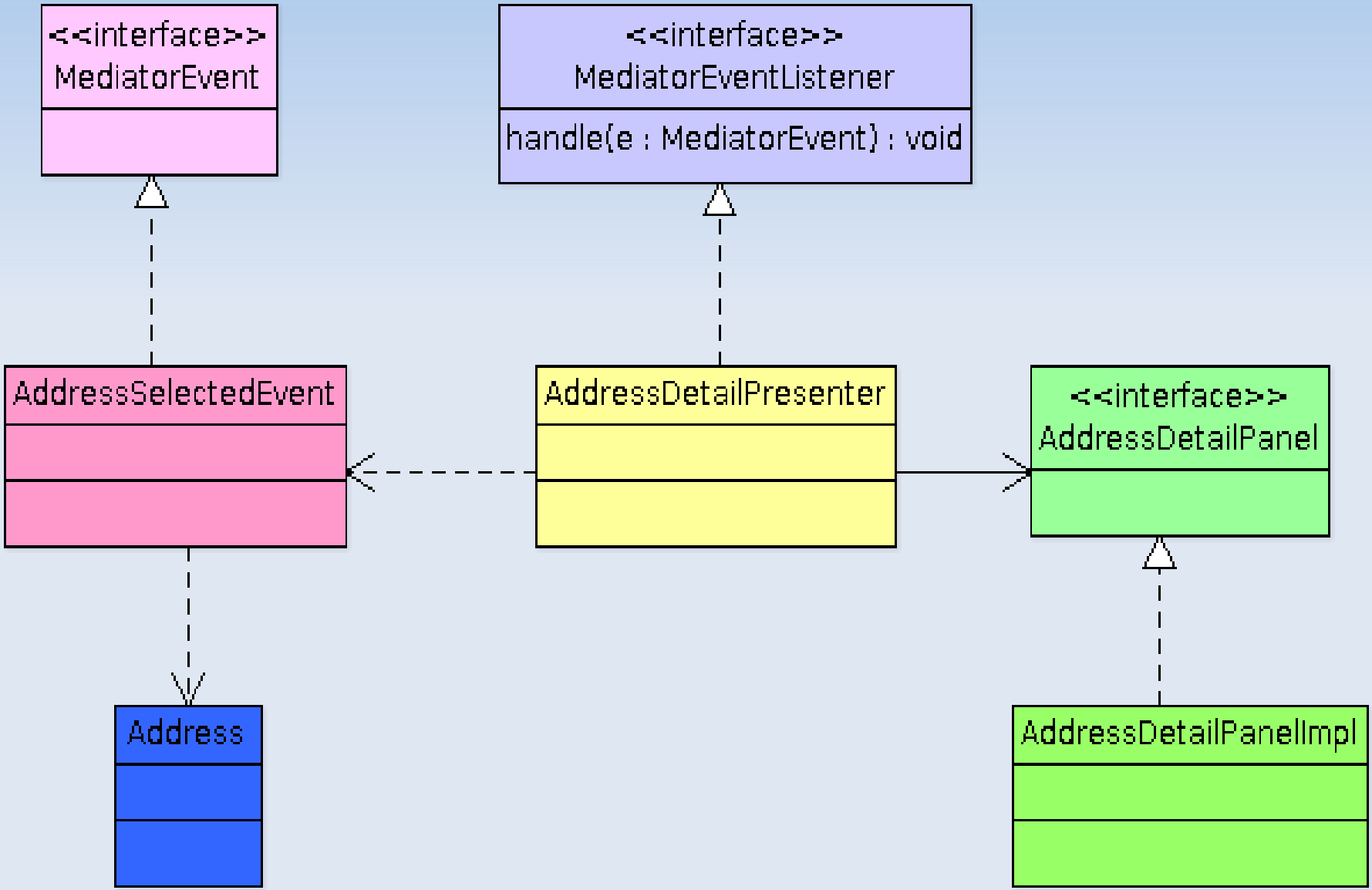



1
registration to the mediator
2
UI event trigger
3
firing business event
4
business event notification
selecting an item causes
a UI event
ListView
updates
UI state
ListPresenter
UI event is translated into
business event and fired
through mediator
registers itself to the mediator
to receive business events
Mediator
notifies about
business event
DetailView
updates
UI state
DetailPresenter
notifies about business event
registers itself to the mediator
to receive business events
UI event is translated into
business event and fired
through mediator
notifies about
business event
registers itself to
the mediator to
receive business
events
clicking a
button causes
a UI event
ToolBar
View
updates
UI state
ToolBar
Presenter

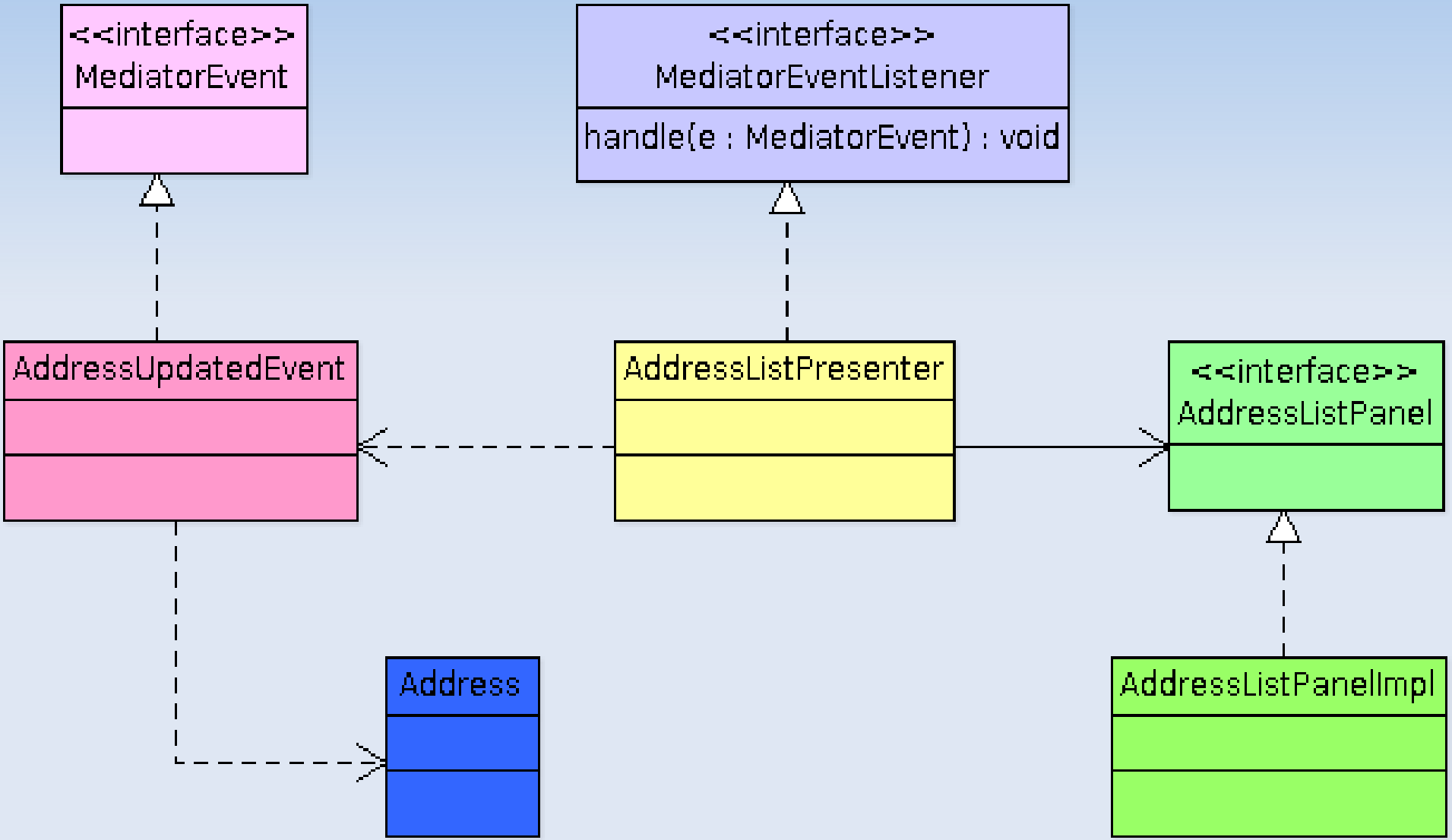
Sınıf ve Arayüzler



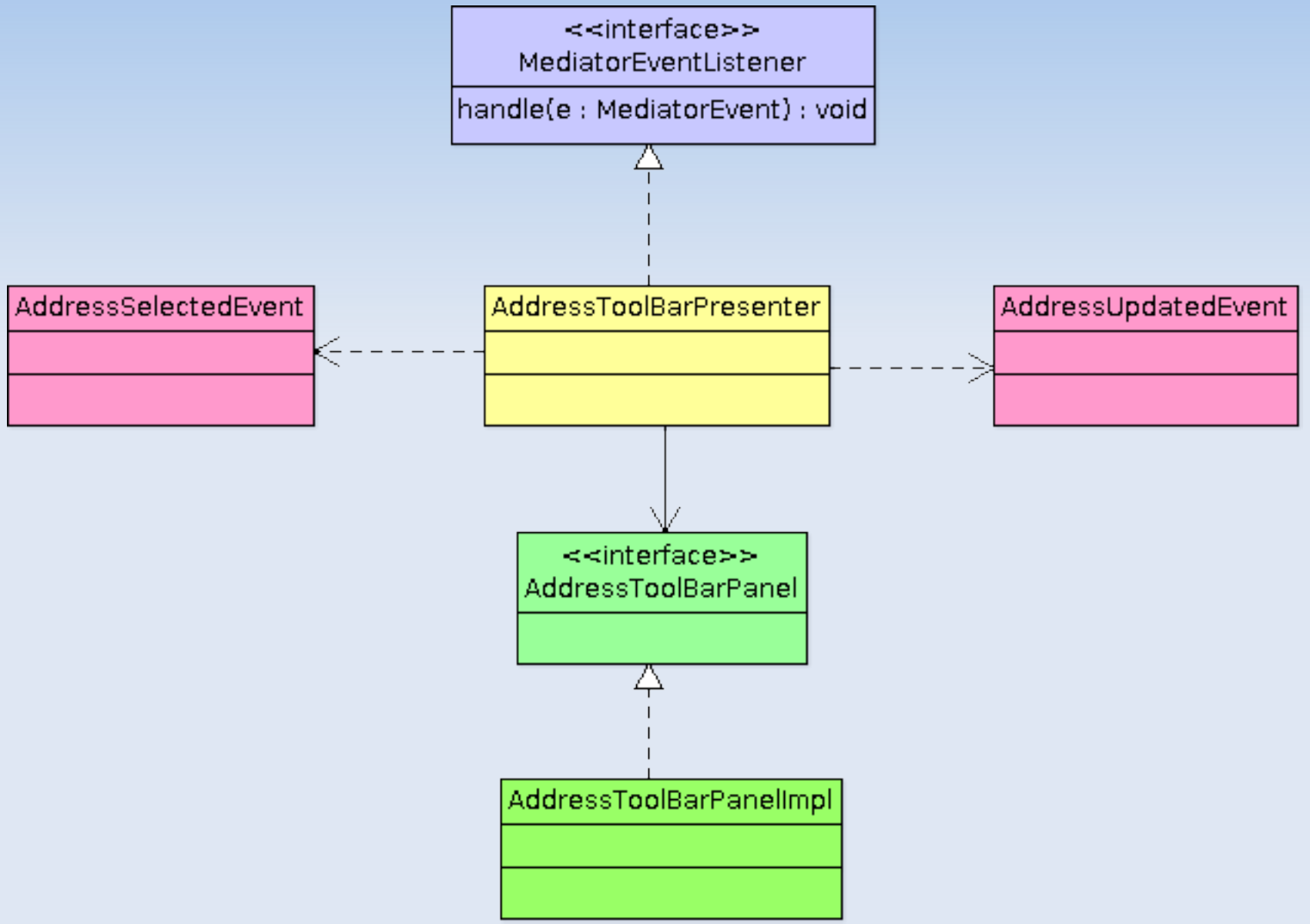
Sınıf ve Arayüzler



Sınıf ve Arayüzler



Sınıf ve Arayüzler



Mediator

```
private Collection<MediatorEventListener> listeners = new
         ArrayList<MediatorEventListener>();

public void addListener(MediatorEventListener listener) {
   listeners.add(listener);
}

public void removeListener(MediatorEventListener listener) {
   listeners.remove(listener);
}

public void fire(MediatorEvent event) {

   for(MediatorEventListener listener:listeners) {

      listener.handle(event);

   }

}
```

Address List Presenter

```
private AddressListPanel listPanel;

public AddressListPresenter(AddressListPanel listPanel,
      AddressService addressService) {

   this.listPanel = listPanel;
   listPanel.loadAddresses(addressService.getAddresses());
}

@Override
public void handle(MediatorEvent event) {

   if(event instanceof AddressUpdatedEvent) {
      AddressUpdatedEvent updateEvent =
(AddressUpdatedEvent)event;
      listPanel.reloadAddress(updateEvent.getAddress());
   }

}
```

Address Detail Presenter

```
private AddressDetailPanel detailPanel;

public AddressDetailPresenter(AddressDetailPanel detailPanel)
{
   this.detailPanel = detailPanel;
}

@Override
public void handle(MediatorEvent event) {

   if(event instanceof AddressSelectedEvent) {

      AddressSelectedEvent selectedEvent =
                     (AddressSelectedEvent)event;

      detailPanel.displayAddress(
            selectedEvent.getSelectedAddress());

   }
}
```

Address ToolBar Presenter

```
private AddressToolBarPanel addressToolBarPanel;

public AddressToolBarPresenter(AddressToolBarPanel
addressToolBarPanel) {
   this.addressToolBarPanel = addressToolBarPanel;
   addressToolBarPanel.swithToSelectionMode();
}

@Override
public void handle(MediatorEvent event) {
   if(event instanceof AddressSelectedEvent) {
      addressToolBarPanel.swithToUpdateMode();

      addressToolBarPanel.setAddress(
         ((AddressSelectedEvent)event).getSelectedAddress());
   } else if(event instanceof AddressUpdatedEvent) {

      addressToolBarPanel.swithToSelectionMode();
   }
}
```

Address List View

```
public AddressListPanelImpl(Mediator mediator) {
   this.mediator = mediator;

...
}

@Override
public void loadAddresses(Collection<Address> addresses) {
...
}

@Override
public void valueChange(ValueChangeEvent event) {
   Address address = (Address) table.getValue();

   AddressSelectedEvent selectedEvent = new
                     AddressSelectedEvent(address);
   mediator.fire(selectedEvent);
}
```

Address ToolBar View

```
public AddressToolBarPanelImpl(Mediator mediator) {
   this.mediator = mediator;
...
}

@Override
public void setAddress(Address address) {
   this.address = address;
}

@Override
public void buttonClick(ClickEvent event) {

   if(event.getButton() == updateButton) {
      mediator.fire(new AddressUpdatedEvent(address));
   }

}
```

Sonuç

- MVC örüntüsü mimarisel olarak sistemi işlevsel olarak birbirinden ayırmaktadır.
- Ancak kullanıcı etkileşimlerinin fonksiyonel davranışa nasıl dönüştürüleceği ile ilgili bir yol göstermemektedir.
- MVP, kullanıcı arayüzünün gösterimi ile fonksiyonel davranışların birbirlerinden bağımsız biçimde ele alınabilmesini sağlamaktadır.

Sonuç

- Davranışların tamamen test edilebilir biçimde geliştirilmesi mümkün hale gelmektedir.
- Arayüz ve iş mantığının geliştirimi tamamen birbirinden ayrılabilmekte ve farklı ekipler tarafından yürütülebilmektedir.
- Mediator örüntüsü ile birbirleri arasında etkileşim ihtiyacı olan bileşenlerin modüler ve birbirlerinden bağımsız biçimde geliştirilmeleri mümkün hale gelmektedir.

İletişim

- Harezmi Bilişim Çözümleri AŞ
- http://www.harezmi.com.tr
- info@harezmi.com.tr

- Kurumsal Java Eğitimleri
- http://www.java-egitimleri.com
- info@java-egitimleri.com